Homework 01 Report

1. Introduction

The goal of this assignment is to build an Agentic AI system capable of analyzing multiple supermarket bill images to answer specific financial queries while filtering out irrelevant questions.

To achieve high accuracy and robustness, I implemented a Router-Solver Architecture using LangChain. This approach decouples natural language understanding (classification) from visual information extraction and uses Python for deterministic calculation. This design specifically addresses the "hallucination" problem common in Multimodal LLMs when performing arithmetic.

2. System Architecture

My implementation consists of three distinct components coordinated by a master_coordinator function.

2.1 Intent Classification (The Router)

The first component is the classification_chain. It only analyzes the user's text query without looking at any images and utilizes a strictly defined system prompt to categorize user input into three specific types:

- Type A: Queries about the total amount paid.
- Type B: Queries about the total amount before discounts.
- Type C: Irrelevant or out-of-scope questions.

Design Logic: To ensure deterministic control flow, the classifier is instructed to output a single character ("A", "B", or "C"). I used StrOutputParser followed by string manipulation (strip().upper()) to sanitize the output. This allows the master coordinator to branch logic immediately: if "C" is detected, the system returns a refusal message without incurring the cost of image processing, otherwise, the system will move on to the next stage.

2.2 Structural Information Extraction

If the query is Type A or Type B, the extraction_chain is triggered. Instead of asking the LLM to perform math, I designed the prompt to function as a parser.

- Prompt Strategy: The system prompt explicitly instructs the model to extract data into a JSON structure containing lists for items and discounts, and a field for receipt_total_paid.
- Handling Edge Cases: A specific rule—"ROUNDING also belongs to discount"—was added to the prompt to ensure that small adjustments in bill totals are correctly categorized as discounts, which is crucial for accurate "Original Price" calculations.

- Batch Processing: The extraction_chain.batch method is used to process multiple image URLs in parallel, returning a list of JSON objects (one per receipt).

2.3 Deterministic Calculation (Python Handlers)

The extracted data is passed to specific Python functions based on the classification result. This ensures mathematical correctness.

- Query 1 Handler (type_a_handler): This function iterates through the extracted JSON list. It sums the receipt_total_paid field from each receipt.

$$Total \quad Spent = \sum_{i=1}^{n} receipt\_total\_paid_i$$

- Query 2 Handler (type_b_handler): This function calculates the original price by adding the paid amount and the sum of all extracted discounts (including rounding adjustments).

$$Original \quad Price = \sum_{i=1}^{n} (receipt\_total\_paid_i + \sum discounts_i)$$

  The logic specifically iterates through the discounts list in the JSON output to aggregate the discount_value.

3. Implementation Details

The system is implemented using Python 3 and the LangChain framework, leveraging Google Gemini Pro Vision as the core reasoning engine. The implementation focuses on modularity and deterministic output control.

3.1 Framework and Model Configuration

Orchestration: I utilized LangChain Expression Language to construct runnable chains. This allows for seamless piping of prompts to the model and then to output parsers.

Model Parameters: The Gemini Pro Vision model(gemini-2.5-flash) is invoked via the VERTEX API. To ensure consistency, the temperature is set to a low value (implicitly controlled via the deterministic nature of the prompt constraints) to minimize creative hallucinations in data extraction.

3.2 Prompt Engineering Strategy

A key implementation detail is the strict constraint design in the system prompts:

For Classification: The prompt enforces a "Single Token Output" rule ("Only output a single capital letter: A, B, or C"). This minimizes parsing errors and allows the StrOutputParser to reliably pass the decision to the control logic without complex regex matching.

For Extraction (JSON Mode): The vision agent is forced to output strictly valid JSON using JsonOutputParser. A critical instruction—"Extract only information explicitly visible"—was implemented to prevent the model from inferring or guessing obscured prices. Furthermore, the prompt explicitly maps "ROUNDING" adjustments to the discount category to ensure mathematical consistency in "Original Price" calculations.

3.3 Deterministic Calculation Logic

To guarantee zero arithmetic errors, all calculations are offloaded to Python functions (type_a_handler and type_b_handler) rather than relying on the LLM's internal math capabilities.

Null Safety: The Python handlers implement robust error checking to handle cases where the model returns null for unclear fields, preventing runtime crashes.

Original Price Reconstruction: The logic for Query 2 actively iterates through the extracted discounts list. It mathematically reconstructs the original price using the formula: $Price_{original} = Price_{paid} + \sum(Discounts)$, ensuring that even complex receipts with multiple discount lines are processed correctly.

3.4 Orchestration Flow

The master_coordinator function serves as the central controller. It employs conditional routing based on the classifier's output.

Efficiency Optimization: If the intent is classified as "C" (Irrelevant), the function returns immediately. This "early exit" strategy prevents the costly execution of the vision extraction chain (extraction_chain.batch), significantly optimizing token usage and latency for out-of-domain queries.

4. Evaluation

The system was tested with the provided evaluation notebook.

Rejection Capability: When presented with irrelevant queries (Type C), the classifier correctly identified them and returned the pre-defined refusal string "I cannot answer this question," complying with the requirement to reject out-of-domain queries.

Arithmetic Accuracy: For Query 1, the system successfully extracted the final total from the bill footer. For Query 2, the extraction agent correctly identified distinct discount lines (and rounding), and the Python handler successfully reconstructed the pre-discount total.

Multi-Image Support: The batch processing logic functioned correctly, aggregating totals across multiple receipt images when provided in a single list.

5. Conclusion

By decoupling the "reading" task (Vision LLM) from the "reasoning" task (Classifier) and the "calculating" task (Python), the implemented agent effectively addresses the homework requirements. The use of structured JSON output combined with Python-based logic ensures that the system is robust against the calculation errors typically associated with pure LLM approaches.